An appendix below shows amendments to the application.

Applicants thank the Examiner for his careful attention to detail in noticing several minor errors in the application.

Drawing margin correction. A formal drawing having the correct margins is submitted for the sheet with FIGS. 12 and 13.

Drawing reference number change. FIG. 11 is changed such that chipset 226 is changed to chipset 227. FIG. 15 is changed such that for coefficient selection 236 is changed to 264 and for strength parameter 238 is changed to 266.

**Drawings showing the changes in red ink as well as corrected formal drawings are provided for the Examiner's approval.**

Objections to Drawings. The following are responses to the objects to the figures.

A) (1) In FIG. 11, the disc is shown as disc 230 and in the paragraph at page 15, line 19, to page 16, line 2, disc 226 is changed to disc 230.

A) (2) The chipset 226 in FIG. 11 is changed to "227." The paragraph at page 15, line 19, to page 16, line 2 is amended accordingly.

A) (3) In FIG. 15, coefficient selection 236 is changed to "264" and strength parameter 238 is changed to "266." The paragraph at page 20, lines 14-18, is amended accordingly.

Objections to the disclosure. The following are responses to the objects to the disclosure.

A) (1) (a) The paragraph at page 7, lines 6-15 is amended to include reference to watermark 100.

A) (1) (b) The paragraph at page 15, lines 19 to page 16, line 2 is amended to include reference to disc 230.

A) (1) (c) The paragraph at page 21, lines 12-21, is amended to include a reference to mechanism 316.

A) (1) (d) The paragraph at page 21, lines 22-28, is amended to include a reference to mechanisms 328 and 336.

B) (1) the paragraph at page 8, lines 12-17, is amended to replace mechanism 102 with mechanism 120.

B) (2) the paragraph at page 15, line 19, to page 16, line 2, is amended to replace memory 222 with memory 224.

B) (3) the paragraph at page 15, line 19, to page 16, line 2, is amended to replace disc 228 with disc 230.

35 U.S.C. § 112, ¶2. Claim 10 stands rejected under 35 U.S.C. § 112, ¶2, as being indefinite. Claim 10 is amended to depend from claim 9.

35 U.S.C. § 102(b): Thomas et al. Claims 1, 9, 11, 12, 19, 24, 25, 26, and 28 stand rejected under 35 U.S.C. § 102(b) as being anticipated by Thomas et al (5,425,100).

It is believed that the present invention and Thomas et al. are significantly different. The present invention concern protecting content. For example, claim 1 recites:

Claim 1 recites:

> "selecting a set of segments of content from a group of segments to be protected; protecting the segments of the set with protection that can be undone; and transmitting the group of segments." (Emphasis added.)

Claims 9, 11, 12, 19, 24, 25, 26, and 28 also include reference to protected segments.

A context into which the present invention may reside is explained in the application on page 5, lines 6-12:

"The invention concerns partially protecting content to be provided to remote computers, only some of which will have the ability and permission to undo the partial protection and produce the entire content remotely. There are a variety of reasons to partial protect content and allow restricted undoing of the protection. For example, under one use, the invention includes placing vacation videos on the World Wide Web, but protecting some segments, such as those showing children. Then, certain family members or friends can see all segments, while other members of the public can see only the undo protection of segments."

By contrast, Thomas et al. is concerned with a way "to measure television (TV) ratings." (Col. 1, line 18.) To accomplish this purpose, the invention relates "to a method and apparatus for monitoring broadcast signals, and more particularly to a universal broadcast code, methods and apparatus for encoding and monitoring a signal." (Col. 1, lines 11-14.) "In brief, the objects and advantages of the present invention are achieved by a multi-level encoded signal monitoring system and a universal broadcast code (UBC). A plurality of encoders are provided for encoding a predetermined program source signal. The program source signal has a plurality of sequential segments. Each encoder is arranged for selectively encoding information on unique specified segments." (Col. 1, line 63 to col. 2, line 2.)

There is nothing in Thomas et al. that suggests that encoding provides protection to the

group of segments. Indeed, many types of encoding provide no protection because decoding is readily understood. For example, MPEG provides encoding, but does not provide protection because decoding of MPEG signals is readily known.

In the absence of any teaching of Thomas et al. that the segments are protected, it is believed that the rejections should be withdrawn.

35 U.S.C. § 103(a): Thomas et al. Claims 2-8, 10, 13-18, 20-23, 27, and 29 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Thomas et al (5,425,100) as applied to claims 1, 9, 11, 12, 19, 24, 25, 26, and 28 above and further in view of obvious variations.

Claims 2-8, 10, 13-18, 20-23, 27, and 29 are each dependent on one of the claims rejected under section 102(b) above. Accordingly, these claims also include reference to protected segments. Because Thomas et al. is significantly different than the present invention, it is believed that the rejections should be withdrawn.

Standards. The Office action (p. 8) states "To complete the record, applicant must supply the standards mentioned at: (a) page 14, line 16-23, "The invention is not ... than MPEG may be used."; and (b) page 19, lines 15-27, "The key may be used in ... of coefficients for sign inversion." Although the necessity of this requirement is traversed, copies of information about Secure Hash Algorithm (SHA) and Message Digest 5 (MD5) are attached to this amendment. Applicants are willing to provide copies of the MPEG standards, however, the undersigned attorney understands that would include several hundred pages. In light of the length of the MPEG standards and how they are readily available in books and on the internet, it seemed to be not useful for applicants to mail a copy of this to the Patent and Trademark Office.

The copy of the information about Secure Hash Algorithm (SHA), "Secure Hash Standard" (1995 April 17), 17 pages, provided with this amendment was obtained from NIST federal standard FIPS 180-1 at http://www.itl.nist.gov/fipspubs/fip180-1.htm.

The copy of the information about Message Digest 5 (MD5), "The Message-Digest Algorithm" (April 1992), 17 pages, provided with this amendment was obtained from RFC 1321 at http://www.faqs.org/rfcs/rfc1321.html.

Conclusion.

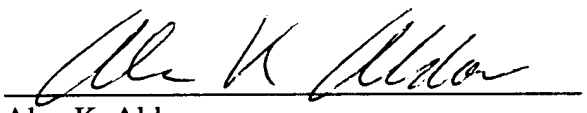
It is believed that the rejections should be withdrawn. Note that merely because applicants do not specifically argue that certain limitations of a claim are not in the references is

not a concession that a reference or combination of references includes the limitations. That applicants do not contradict a particular statement made in the Office action is not a concession that applicant agrees with it. Further, merely because applicants do not separately argue the patentability of every dependent claim is not a concession that there are not additional reasons for patentability of these dependent claims.

Applicant believes the application is in condition for allowance and respectfully requests the same.

Respectfully submitted,

Dated: March 1, 2002



Alan K. Aldous
Reg. No. 31,905

Blakely, Sokoloff, Taylor & Zafman
12400 Wilshire Boulevard, Seventh Floor
Los Angeles, California 90025-1026
Phone: (503) 264-7125
Phone: (503) 684-6200
Phone (310) 207-3800
Facsimile: (503) 684-3245

Amendments to the application are shown in an Appendix beginning on the following page.

APPENDIX:

Please amend the application as follows.

VERSION WITH MARKINGS TO SHOW AMENDMENTS

Amend the paragraph at page 7, lines 6-15, as follows:

A window 80 includes a display 84 for displaying one of the segments, which may be selected, paused or stopped through icons 90 or other means. A scroll bar 82 may be used to advance through frames of the segment selected for viewing in display 84. The various icons described herein can be activated through a mouse. Activation of a browse icon 92 may cause segment in display 66 to also appear in display 84. Bit encryption and visual scrambling selection boxes 94 and 96 can be checked with a click of a mouse to select bit encryption and/or visual scrambling features described below. In some embodiments, when either of these boxes is checked, the corresponding display in window 64 is enclosed in a rectangle or otherwise designated as being protected. The protection occurs in response to encode icon 98 being activated with a click of a mouse. For example, display 68 and 74 are enclosed in a rectangle indicating that segments 2 and 5 (which include images I2 and I5) will be protected if encode icon 98 is activated. Activation of a watermark icon 100 causes information such as is described below to be contained in a watermark.

Amend the paragraph at page 8, lines 12-17, as follows:

FIG. 4 illustrates a content providing system 114 which is similar to content providing system 14 but illustrates some additional capabilities, which could be included in content providing system 14. A segment creation mechanism 120 represents a user interface and associated software to select segments of the group of segments (e.g., to designate the beginning and ending frames or time of the segment). Mechanism [102] 120 may be used for joining disjointed segments in a group and/or dividing continuous content into segments of a group.

Amend the paragraph at page 15, line 19, to page 16, line 2, as follows:

FIG. 11 illustrates a computer 220 (which may be an example of system 14) including a processor 222, on-die memory 224, chipset I/O [226] 227, and off-die memory 228. Memory [222] 224, memory 228, and a disc [228] 230 include machine readable media to hold

instructions to be executed and other data. The various block diagram and flow chart blocks in the other figures called mechanisms may represent processor 222 performing functions on software or may represent hardware other than processor 222 performing the functions described in connection with the block diagram or flowchart mechanisms. A link 234 joins computer 220 to a remote computer 236 (which may be an example of remote receiving computer 20). Computer 236 may be the same as of different than computer 220. A display 238 may be packaged with or separate from computer 236. Link 234 represents any of various links including the Internet, an intranet, a local area network, satellite, or other networks. The term computer is intended to be broadly interpreted to include a variety of systems and devices including personal computers, mainframe computers, set top boxes, digital versatile disc (DVD) players, and the like.

Amend the paragraph at page 20, lines 14-18, as follows:

The invention may be used with respect to signals not previously compressed. FIG. 15 illustrates an encode mechanism 270 in which uncompressed (raw) video is first transformed with a DCT mechanism 272 (which may be the same as encoder 200 in FIG. 9). Scrambling mechanism 244 alters the coefficients as described above. An inverse DCT mechanism 276 returns the scrambled video to the uncompressed (raw) video format. Selected coefficients are provided by coefficient selection mechanism 264 responsive to a key and strength parameter 266.

Amend the paragraph at page 21, lines 12-21, as follows:

As an example, FIG. 17 illustrates a scrambling encode mechanism 300 (which may be in computer 220 in FIG. 11) in which video blocks (which may be in MPEG format) are received by in buffer 302. In some embodiments, as a block is received, it is identified with a number m or placed in position m of the buffer. The number m is incremented by increment mechanism 308 with each received block until m = N (compare mechanism 306), where N is the number of blocks available for permutation. For example, if a set of four blocks may be permuted, N is 3 (assuming m starts at 0). When m = N, order selection mechanism 312 selects a block order based on a key and sets m to 0 (mechanism 316). The blocks are read from buffer 302 in the permuted block order as specified in the block order from order selection mechanism 312. The

block order may be a mapping for each block, wherein or not it is changed or only those that change order.

Amend the paragraph at page 21, lines 22-28, as follows:

FIG. 18 illustrates a descrambling decode mechanism 320 (which may be in computer 236 in FIG. 11) which receives the blocks in permuted order in buffer 322 from buffer 302 in FIG. 17. When the buffer is full (comparison mechanism 326), order selection mechanism 332 selects the block order responsive to a key and buffer 322. Responsive to the block order, the blocks in the original order are read from buffer 322 in the original order. Mechanism 328 increments m and mechanism 336 sets m = 0. By using the same block order as in FIG. 17, an inverse permutation occurs and the blocks are read out in the original order.

Please amend claim 10 as follows:

10. (Amended) The method of claim [8] 9, wherein the video signals are in an MPEG format.

COPY OF PAPERS
ORIGINALLY FILED

Return to the FIPS
Home Page

**FIPS PUB 180-1**
Supersedes FIPS PUB 180
1993 May 11




Federal Information
Processing Standards Publication 180-1

1995 April 17
**Announcing the Standard for**

# SECURE HASH STANDARD

(The Foreword, Abstract, and Key Words
can be found at the end of this document.)

**Federal Information Processing Standards Publications (FIPS PUBS) are issued by the National Institute of Standards and Technology after approval by the Secretary of Commerce pursuant to Section 111(d) of the Federal Property and Administrative Services Act of 1949, as amended by the Computer Security Act of 1987, Public Law 100-235.**

**Name of Standard:** Secure Hash Standard.

**Category of Standard:** Computer Security.

**Explanation:** This Standard specifies a Secure Hash Algorithm, SHA-1, for computing a condensed representation of a message or a data file. When a message of any length $< 2^{64}$ bits is input, the SHA-1 produces a 160-bit output called a message digest. The message digest can then be input to the Digital Signature Algorithm (DSA) which generates or verifies the signature for the message. Signing the message digest rather than the message often improves the efficiency of the process because the message digest is usually much smaller in size than the message. The same hash algorithm must be used by the verifier of a digital signature as was used by the creator of the digital signature.

The SHA-1 is called secure because it is computationally infeasible to find a message which corresponds to a given message digest, or to find two different messages which produce the same message digest. Any change to a message in transit will, with very high probability, result in a different message digest, and the signature will fail to verify. SHA-1 is a technical revision of SHA (FIPS 180). A circular left shift operation has been added to the specifications in section 7, line b, page 9 of FIPS 180 and its equivalent in section 8, line c, page 10 of FIPS 180. This revision improves the security provided by this standard. The SHA-1 is based on principles similar to those used by Professor Ronald L. Rivest of MIT when designing the MD4 message digest algorithm ("The MD4 Message Digest Algorithm," Advances in Cryptology - CRYPTO '90 Proceedings, Springer-Verlag, 1991, pp. 303-311), and is closely modelled after that algorithm.

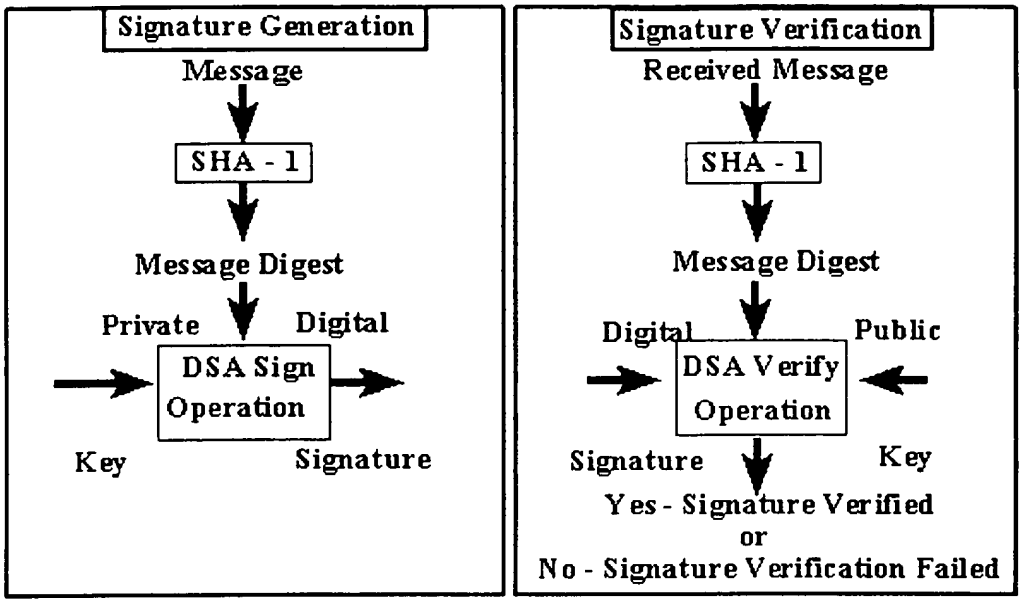



**Figure 1: Using the SHA-1 with the DSA**

**Approving Authority:** Secretary of Commerce.

**Maintenance Agency:** U.S. Department of Commerce, National Institute of Standards and Technology, Computer Systems Laboratory.

**Applicability:** This standard is applicable to all Federal departments and agencies for the protection of unclassified information that is not subject to section 2315 of Title 10, United States Code, or section 3502(2) of Title 44, United States Code. This standard is required for use with the Digital Signature Algorithm (DSA) as specified in the Digital Signature Standard (DSS) and whenever a secure hash algorithm is required for Federal applica- tions. Private and commercial organizations are encouraged to adopt and use this standard.

**Applications:** The SHA-1 may be used with the DSA in electronic mail, electronic funds transfer, software distribution, data storage, and other applications which require data integrity assurance and data origin authentication. The SHA-1 may also be used whenever it is necessary to generate a condensed version of a message.

**Implementations:** The SHA-1 may be implemented in software, firmware, hardware, or any combination thereof. Only implementations of the SHA-1 that are validated by NIST will be considered as complying with this standard. Information about the requirements for validating implementations of this standard can be obtained from the National Institute of Standards and Technology, Computer Systems Laboratory, Attn: SHS Validation, Gaithersburg, MD 20899.

**Export Control:** Implementations of this standard are subject to FederalGovernment export controls as specified in Title 15, Code of Federal Regulations, Parts 768 through 799. Exporters are advised to contact the Department of Commerce, Bureau of Export Administration for more information.

**Patents:** Implementations of the SHA-1 in this standard may be covered by U.S. and foreign patents.

**Implementation Schedule:** This standard becomes effective October 2, 1995.

**Specifications:** Federal Information Processing Standard (FIPS 180-1) Secure Hash Standard (affixed).

**Cross Index:**

a. FIPS PUB 46-2, Data Encryption Standard.

b. FIPS PUB 73, Guidelines for Security of Computer Applications.

c. FIPS PUB 140-1, Security Requirements for Cryptographic Modules.

d. FIPS PUB 186, Digital Signature Standard.

e. Federal Informations Resources Management Regulations (FIRMR) subpart 201.20.303, Standards, and subpart 201.39.1002, Federal Standards.

**Objectives:** The objectives of this standard are to:

a. Specify the secure hash algorithm required for use with the Digital Signature Standard (FIPS 186) in the generation and verification of digital signatures;

b. Specify the secure hash algorithm to be used whenever a secure hash algorithm is required for Federal applications; and

c. Encourage the adoption and use of the specified secure hash algorithm by private and commercial organizations.

**Qualifications:** While it is the intent of this standard to specify a secure hash algorithm, conformance to this standard does not assure that a particular implementation is secure. The responsible authority in each agency or department shall assure that an overall implementation provides an acceptable level of security. This standard will be reviewed every five years in order to assess its adequacy.

**Waiver Procedure:** Under certain exceptional circumstances, the heads of Federal departments and agencies may approve waivers to Federal Information Processing Standards (FIPS). The head of such agency may redelegate such authority only to a senior official designated pursuant to section 3506(b) of Title 44, United States Code. Waiver shall be granted only when:

a. Compliance with a standard would adversely affect the accomplishment of the mission of an operator of a Federal computer system; or

b. Compliance with a standard would cause a major adverse financial impact on the operator which is not offset by Government-wide savings.

Agency heads may act upon a written waiver request containing the information detailed above.

Agency heads may also act without a written waiver request when they determine that conditions for meeting the standard cannot be met. Agency heads may approve waivers only by a written decision which explains the basis on which the agency head made the required finding(s). A copy of each decision, with procurement sensitive or classified portions clearly identified, shall be sent to: National Institute of Standards and Technology; ATTN: FIPS Waiver Decisions, Technology Building, Room B-154, Gaithersburg, MD 20899.

In addition, notice of each waiver granted and each delegation of authority to approve waivers shall be sent promptly to the Committee on Government Operations of the House of Representatives and the Committee on Government Affairs of the Senate and shall be published promptly in the Federal Register.

When the determination on a waiver applies to the procurement of equipment and/or services, a notice of the waiver determination must be published in the Commerce Business Daily as a part of the notice of solicitation for offers of an acquisition or, if the waiver determination is made after that notice is published, by amendment to such notice.

A copy of the waiver, any supporting documents, the document approving the waiver and any accompanying documents, with such deletions as the agency is authorized and decides to make under 5 United States Code Section 552(b), shall be part of the procurement documentation and retained by the agency.

**Where to Obtain Copies of the Standard:** Copies of this publication are for sale by the National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161. When ordering, refer to Federal Information Processing Standards Publication 180-1 (FIPSPUB180-1), and identify the title. When microfiche is desired, this should be specified. Prices are published by NTIS in current catalogs and other issuances. Payment may be made by check, money order, deposit account or charged to a credit card accepted by NTIS.

---

**FIPS PUB 180-1**
Supersedes FIPS PUB 180
1993 May 11

Federal Information
Processing Standards Publication 180-1

1995 April 17
**Specifications for**

# SECURE HASH STANDARD

## 1. INTRODUCTION

The Secure Hash Algorithm (SHA-1) is required for use with the Digital Signature Algorithm

(DSA) as specified in the Digital Signature Standard (DSS) and whenever a secure hash algorithm is required for federal applica- tions. For a message of length < 2^64 bits, the SHA-1 produces a 160-bit condensed representation of the message called a message digest. The message digest is used during generation of a signature for the message. The SHA-1 is also used to compute a message digest for the received version of the message during the process of verifying the signature. Any change to the message in transit will, with very high probability, result in a different message digest, and the signature will fail to verify.

The SHA-1 is designed to have the following properties: it is computationally infeasible to find a message which corresponds to a given message digest, or to find two different messages which produce the same message digest.

## 2. BIT STRINGS AND INTEGERS

The following terminology related to bit strings and integers will be used:

a. A hex digit is an element of the set {0, 1, ... , 9, A, ... , F}. A hex digit is the representation of a 4-bit string. **Examples:** 7 = 0111, A = 1010.

b. A word equals a 32-bit string which may be represented as a sequence of 8 hex digits. To convert a word to 8 hex digits each 4-bit string is converted to its hex equivalent as described in (a) above. **Example:**

1010 0001 0000 0011 1111 1110 0010 0011 = A103FE23.

c. An integer between 0 and $2^{32}$ - 1 inclusive may be represented as a word. The least significant four bits of the integer are represented by the right-most hex digit of the word representation. **Example:** the integer 291 = $2^8+2^5+2^1+2^0$ = 256+32+2+1 is represented by the hex word, 00000123.

If z is an integer, 0 <= z < $2^{64}$, then z = $2^{32}$x + y where 0 <= x < $2^{32}$ and 0 <= y < $2^{32}$. Since x and y can be represented as words X and Y, respectively, z can be represented as the pair of words (X,Y).

d. block = 512-bit string. A block (e.g., B) may be represented as a sequence of 16 words.

## 3. OPERATIONS ON WORDS

The following logical operators will be applied to words:

a. Bitwise logical word operations

```
X ^ Y          =  bitwise logical "and" of  X and Y.

X \/ Y         =  bitwise logical "inclusive-or" of X and Y.

X XOR Y        =  bitwise logical "exclusive-or" of X and Y.

~ X            =  bitwise logical "complement" of X.
```

**Example:**

```
        01101100101110011101001001111011
XOR     01100101110000010110100110110111
        --------------------------------
  =     00001001011110001011101111001100
```

b. The operation X + Y is defined as follows: words X and Y represent integers x and y, where $0 <= x < 2^{32}$ and $0 <= y < 2^{32}$. For positive integers n and m, let n mod m be the remainder upon dividing n by m. Compute

$z = (x + y) \bmod 2^{32}$.

Then $0 <= z < 2^{32}$. Convert z to a word, Z, and define Z = X + Y.

c. The circular left shift operation $S^n(X)$, where X is a word and n is an integer with $0 <= n < 32$, is defined by

$S^n(X) = (X << n) \text{ OR } (X >> 32-n)$.

In the above, X << n is obtained as follows: discard the left-most n bits of X and then pad the result with n zeroes on the right (the result will still be 32 bits). X >> n is obtained by discarding the right-most n bits of X and then padding the result with n zeroes on the left. Thus $S^n(X)$ is equivalent to a circular shift of X by n positions to the left.

## 4. MESSAGE PADDING

The SHA-1 is used to compute a message digest for a message or data file that is provided as input. The message or data file should be considered to be a bit string. The length of the message is the number of bits in the message (the empty message has length 0). If the number of bits in a message is a multiple of 8, for compactness we can represent the message in hex. The purpose of message padding is to make the total length of a padded message a multiple of 512. The SHA-1 sequentially processes blocks of 512 bits when computing the message digest. The following specifies how this padding shall be performed. As a summary, a "1" followed by m "0"s followed by a 64-bit integer are appended to the end of the message to produce a padded message of length 512 * n. The 64-bit integer is l, the length of the original message. The padded message is then processed by the SHA-1 as n 512-bit blocks.

Suppose a message has length $l < 2^{64}$. Before it is input to the SHA-1, the message is padded on the right as follows:

a. "1" is appended. **Example:** if the original message is "01010000", this is padded to "010100001".

b. "0"s are appended. The number of "0"s will depend on the original length of the message. The last 64 bits of the last 512-bit block are reserved for the length l of the original message.

**Example:** Suppose the original message is the bit string

01100001 01100010 01100011 01100100 01100101.

After step (a) this gives

01100001 01100010 01100011 01100100 01100101 1.

Since l = 40, the number of bits in the above is 41 and 407 "0"s are appended, making the total now 448. This gives (in hex)

61626364 65800000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000.

c. Obtain the 2-word representation of l, the number of bits in the original message. If $l < 2^{32}$ then the first word is all zeroes. Append these two words to the padded message.

**Example:** Suppose the original message is as in (b). Then l = 40 (note that l is computed before any padding). The two-word representation of 40 is hex 00000000 00000028. Hence the final padded message is hex

61626364 65800000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000028.

The padded message will contain 16 * n words for some n > 0. The padded message is regarded as a sequence of n blocks $M_1$ , $M_2$, ... , $M_n$, where each $M_i$ contains 16 words and $M_1$ contains the first characters (or bits) of the message.

## 5. FUNCTIONS USED

A sequence of logical functions $f_0$, $f_1$,..., $f_{79}$ is used in the SHA-1. Each $f_t$, 0 <= t <= 79, operates on three 32-bit words B, C, D and produces a 32-bit word as output. $f_t$ (B,C,D) is defined as follows: for words B, C, D,

$f_t$(B,C,D) = (B AND C) OR ((NOT B) AND D) ( 0 <= t <= 19)

$f_t$(B,C,D) = B XOR C XOR D (20 <= t <= 39)

$f_t$(B,C,D) = (B AND C) OR (B AND D) OR (C AND D) (40 <= t <= 59)

$f_t$(B,C,D) = B XOR C XOR D (60 <= t <= 79).

## 6. CONSTANTS USED

A sequence of constant words K(0), K(1), ... , K(79) is used in the SHA-1. In hex these are given by

K = 5A827999 ( 0 <= t <= 19)

$K_t$ = 6ED9EBA1 (20 <= t <= 39)

$K_t$ = 8F1BBCDC (40 <= t <= 59)

$K_t$ = CA62C1D6 (60 <= t <= 79).

## 7. COMPUTING THE MESSAGE DIGEST

The message digest is computed using the final padded message. The computation uses two buffers, each consisting of five 32-bit words, and a sequence of eighty 32-bit words. The words of the first 5-word buffer are labeled A,B,C,D,E. The words of the second 5-word buffer are labeled $H_0$, $H_1$, $H_2$, $H_3$, $H_4$. The words of the 80-word sequence are labeled $W_0$, $W_1$,..., $W_{79}$. A single word buffer TEMP is also employed.

To generate the message digest, the 16-word blocks $M_1$, $M_2$,..., $M_n$ defined in Section 4 are processed in order. The processing of each $M_i$ involves 80 steps.

Before processing any blocks, the $\{H_i\}$ are initialized as follows: in hex,

$H_0$ = 67452301

$H_1$ = EFCDAB89

$H_2$ = 98BADCFE

$H_3$ = 10325476

$H_4$ = C3D2E1F0.

Now $M_1$, $M_2$, ... , $M_n$ are processed. To process $M_i$, we proceed as follows:

a. Divide $M_i$ into 16 words $W_0$, $W_1$, ... , $W_{15}$, where $W_0$ is the left-most word.

b. For t = 16 to 79 let $W_t = S^1(W_{t-3}$ XOR $W_{t-8}$ XOR $W_{t-14}$ XOR $W_{t-16})$.

c. Let $A = H_0$, $B = H_1$, $C = H_2$, $D = H_3$, $E = H_4$.

d. For t = 0 to 79 do

$TEMP = S^5(A) + f_t(B,C,D) + E + W_t + K_t$;

$E = D$; $D = C$; $C = S^{30}(B)$; $B = A$; $A = TEMP$;

e. Let $H_0 = H_0 + A$, $H_1 = H_1 + B$, $H_2 = H_2 + C$, $H_3 = H_3 + D$, $H_4 = H_4 + E$.

After processing $M_n$, the message digest is the 160-bit string represented by the 5 words

$H_0\ H_1\ H_2\ H_3\ H_4$.

## 8. ALTERNATE METHOD OF COMPUTATION

The above assumes that the sequence $W_0, \ldots, W_{79}$ is implemented as an array of eighty 32-bit words. This is efficient from the standpoint of minimization of execution time, since the addresses of $W_{t-3}, \ldots, W_{t-16}$ in step (b) are easily computed. If space is at a premium, an alternative is to regard { $W_t$ } as a circular queue, which may be implemented using an array of sixteen 32-bit words W[0], ... W[15]. In this case, in hex let MASK = 0000000F. Then processing of $M_i$ is as follows:

a. Divide $M_i$ into 16 words W[0], ... , W[15], where W[0] is the left-most word.

b. Let $A = H_0$, $B = H_1$, $C = H_2$, $D = H_3$, $E = H_4$.

c. For t = 0 to 79 do
s = t ^ MASK;

if (t >= 16) W[s] = $S^1$(W[(s + 13) ^ MASK] XOR W[(s + 8) AND MASK] XOR W[(s + 2) ^ MASK] XOR W[s]);

TEMP = $S^5(A) + f_t(B,C,D) + E + W[s] + K_t$;

E = D; D = C; C = $S^{30}$(B); B = A; A = TEMP;

d. Let $H_0 = H_0 + A$, $H_1 = H_1 + B$, $H_2 = H_2 + C$, $H_3 = H_3 + D$, $H_4 = H_4 + E$.

## 9. COMPARISON OF METHODS

The methods of Sections 7 and 8 yield the same message digest. Although using the method of Section 8 saves sixty-four 32-bit words of storage, it is likely to lengthen execution time due to the increased complexity of the address computations for the { W[t] } in step (c). Other computation methods which give identical results may be implemented in conformance with the standard.

## APPENDIX A. A SAMPLE MESSAGE AND ITS MESSAGE DIGEST

This appendix is for informational purposes only and is not required to meet the standard.

Let the message be the ASCII binary-coded form of "abc", i.e.,
01100001 01100010 01100011.

This message has length l = 24. In step (a) of Section 4, we append "1". In step (b) we append 423 "0"s. In step (c) we append hex 00000000 00000018, the 2-word

representation of 24. Thus the final padded message consists of one block, so that n = 1 in the notation of Section 4.

The initial hex values of $\{H_i\}$ are

$H_0$ = 67452301
$H_1$ = EFCDAB89
$H_2$ = 98BADCFE
$H_3$ = 10325476
$H_4$ = C3D2E1F0.

Start processing block 1. The words of block 1 are

W[0] = 61626380
W[1] = 00000000
W[2] = 00000000
W[3] = 00000000
W[4] = 00000000
W[5] = 00000000
W[6] = 00000000
W[7] = 00000000
W[8] = 00000000
W[9] = 00000000
W[10] = 00000000
W[11] = 00000000
W[12] = 00000000
W[13] = 00000000
W[14] = 00000000
W[15] = 00000018.

The hex values of A,B,C,D,E after pass t of the "for t = 0 to 79" loop (step (d) of Section 7 or step (c) of Section 8) are

| | A | B | C | D | E |
|---|---|---|---|---|---|
| t = 0: | 0116FC33 | 67452301 | 7BF36AE2 | 98BADCFE | 10325476 |
| t = 1: | 8990536D | 0116FC33 | 59D148C0 | 7BF36AE2 | 98BADCFE |
| t = 2: | A1390F08 | 8990536D | C045BF0C | 59D148C0 | 7BF36AE2 |
| t = 3: | CDD8E11B | A1390F08 | 626414DB | C045BF0C | 59D148C0 |
| t = 4: | CFD499DE | CDD8E11B | 284E43C2 | 626414DB | C045BF0C |
| t = 5: | 3FC7CA40 | CFD499DE | F3763846 | 284E43C2 | 626414DB |
| t = 6: | 993E30C1 | 3FC7CA40 | B3F52677 | F3763846 | 284E43C2 |
| t = 7: | 9E8C07D4 | 993E30C1 | 0FF1F290 | B3F52677 | F3763846 |
| t = 8: | 4B6AE328 | 9E8C07D4 | 664F8C30 | 0FF1F290 | B3F52677 |
| t = 9: | 8351F929 | 4B6AE328 | 27A301F5 | 664F8C30 | 0FF1F290 |
| t = 10: | FBDA9E89 | 8351F929 | 12DAB8CA | 27A301F5 | 664F8C30 |
| t = 11: | 63188FE4 | FBDA9E89 | 60D47E4A | 12DAB8CA | 27A301F5 |
| t = 12: | 4607B664 | 63188FE4 | 7EF6A7A2 | 60D47E4A | 12DAB8CA |
| t = 13: | 9128F695 | 4607B664 | 18C623F9 | 7EF6A7A2 | 60D47E4A |
| t = 14: | 196BEE77 | 9128F695 | 1181ED99 | 18C623F9 | 7EF6A7A2 |
| t = 15: | 20BDD62F | 196BEE77 | 644A3DA5 | 1181ED99 | 18C623F9 |
| t = 16: | 4E925823 | 20BDD62F | C65AFB9D | 644A3DA5 | 1181ED99 |

| | | | | | |
|---|---|---|---|---|---|
| t = 17: | 82AA6728 | 4E925823 | C82F758B | C65AFB9D | 644A3DA5 |
| t = 18: | DC64901D | 82AA6728 | D3A49608 | C82F758B | C65AFB9D |
| t = 19: | FD9E1D7D | DC64901D | 20AA99CA | D3A49608 | C82F758B |
| t = 20: | 1A37B0CA | FD9E1D7D | 77192407 | 20AA99CA | D3A49608 |
| t = 21: | 33A23BFC | 1A37B0CA | 7F67875F | 77192407 | 20AA99CA |
| t = 22: | 21283486 | 33A23BFC | 868DEC32 | 7F67875F | 77192407 |
| t = 23: | D541F12D | 21283486 | 0CE88EFF | 868DEC32 | 7F67875F |
| t = 24: | C7567DC6 | D541F12D | 884A0D21 | 0CE88EFF | 868DEC32 |
| t = 25: | 48413BA4 | C7567DC6 | 75507C4B | 884A0D21 | 0CE88EFF |
| t = 26: | BE35FBD5 | 48413BA4 | B1D59F71 | 75507C4B | 884A0D21 |
| t = 27: | 4AA84D97 | BE35FBD5 | 12104EE9 | B1D59F71 | 75507C4B |
| t = 28: | 8370B52E | 4AA84D97 | 6F8D7EF5 | 12104EE9 | B1D59F71 |
| t = 29: | C5FBAF5D | 8370B52E | D2AA1365 | 6F8D7EF5 | 12104EE9 |
| t = 30: | 1267B407 | C5FBAF5D | A0DC2D4B | D2AA1365 | 6F8D7EF5 |
| t = 31: | 3B845D33 | 1267B407 | 717EEBD7 | A0DC2D4B | D2AA1365 |
| t = 32: | 046FAA0A | 3B845D33 | C499ED01 | 717EEBD7 | A0DC2D4B |
| t = 33: | 2C0EBC11 | 046FAA0A | CEE1174C | C499ED01 | 717EEBD7 |
| t = 34: | 21796AD4 | 2C0EBC11 | 811BEA82 | CEE1174C | C499ED01 |
| t = 35: | DCBBB0CB | 21796AD4 | 4B03AF04 | 811BEA82 | CEE1174C |
| t = 36: | 0F511FD8 | DCBBB0CB | 085E5AB5 | 4B03AF04 | 811BEA82 |
| t = 37: | DC63973F | 0F511FD8 | F72EEC32 | 085E5AB5 | 4B03AF04 |
| t = 38: | 4C986405 | DC63973F | 03D447F6 | F72EEC32 | 085E5AB5 |
| t = 39: | 32DE1CBA | 4C986405 | F718E5CF | 03D447F6 | F72EEC32 |
| t = 40: | FC87DEDF | 32DE1CBA | 53261901 | F718E5CF | 03D447F6 |
| t = 41: | 970A0D5C | FC87DEDF | 8CB7872E | 53261901 | F718E5CF |
| t = 42: | 7F193DC5 | 970A0D5C | FF21F7B7 | 8CB7872E | 53261901 |
| t = 43: | EE1B1AAF | 7F193DC5 | 25C28357 | FF21F7B7 | 8CB7872E |
| t = 44: | 40F28E09 | EE1B1AAF | 5FC64F71 | 25C28357 | FF21F7B7 |
| t = 45: | 1C51E1F2 | 40F28E09 | FB86C6AB | 5FC64F71 | 25C28357 |
| t = 46: | A01B846C | 1C51E1F2 | 503CA382 | FB86C6AB | 5FC64F71 |
| t = 47: | BEAD02CA | A01B846C | 8714787C | 503CA382 | FB86C6AB |
| t = 48: | BAF39337 | BEAD02CA | 2806E11B | 8714787C | 503CA382 |
| t = 49: | 120731C5 | BAF39337 | AFAB40B2 | 2806E11B | 8714787C |
| t = 50: | 641DB2CE | 120731C5 | EEBCE4CD | AFAB40B2 | 2806E11B |
| t = 51: | 3847AD66 | 641DB2CE | 4481CC71 | EEBCE4CD | AFAB40B2 |
| t = 52: | E490436D | 3847AD66 | 99076CB3 | 4481CC71 | EEBCE4CD |
| t = 53: | 27E9F1D8 | E490436D | 8E11EB59 | 99076CB3 | 4481CC71 |
| t = 54: | 7B71F76D | 27E9F1D8 | 792410DB | 8E11EB59 | 99076CB3 |
| t = 55: | 5E6456AF | 7B71F76D | 09FA7C76 | 792410DB | 8E11EB59 |
| t = 56: | C846093F | 5E6456AF | 5EDC7DDB | 09FA7C76 | 792410DB |
| t = 57: | D262FF50 | C846093F | D79915AB | 5EDC7DDB | 09FA7C76 |
| t = 58: | 09D785FD | D262FF50 | F211824F | D79915AB | 5EDC7DDB |
| t = 59: | 3F52DE5A | 09D785FD | 3498BFD4 | F211824F | D79915AB |
| t = 60: | D756C147 | 3F52DE5A | 4275E17F | 3498BFD4 | F211824F |
| t = 61: | 548C9CB2 | D756C147 | 8FD4B796 | 4275E17F | 3498BFD4 |
| t = 62: | B66C020B | 548C9CB2 | F5D5B051 | 8FD4B796 | 4275E17F |
| t = 63: | 6B61C9E1 | B66C020B | 9523272C | F5D5B051 | 8FD4B796 |
| t = 64: | 19DFA7AC | 6B61C9E1 | ED9B0082 | 9523272C | F5D5B051 |
| t = 65: | 101655F9 | 19DFA7AC | 5AD87278 | ED9B0082 | 9523272C |
| t = 66: | 0C3DF2B4 | 101655F9 | 0677E9EB | 5AD87278 | ED9B0082 |
| t = 67: | 78DD4D2B | 0C3DF2B4 | 4405957E | 0677E9EB | 5AD87278 |
| t = 68: | 497093C0 | 78DD4D2B | 030F7CAD | 4405957E | 0677E9EB |
| t = 69: | 3F2588C2 | 497093C0 | DE37534A | 030F7CAD | 4405957E |
| t = 70: | C199F8C7 | 3F2588C2 | 125C24F0 | DE37534A | 030F7CAD |
| t = 71: | 39859DE7 | C199F8C7 | 8FC96230 | 125C24F0 | DE37534A |
| t = 72: | EDB42DE4 | 39859DE7 | F0667E31 | 8FC96230 | 125C24F0 |
| t = 73: | 11793F6F | EDB42DE4 | CE616779 | F0667E31 | 8FC96230 |
| t = 74: | 5EE76897 | 11793F6F | 3B6D0B79 | CE616779 | F0667E31 |
| t = 75: | 63F7DAB7 | 5EE76897 | C45E4FDB | 3B6D0B79 | CE616779 |
| t = 76: | A079B7D9 | 63F7DAB7 | D7B9DA25 | C45E4FDB | 3B6D0B79 |
| t = 77: | 860D21CC | A079B7D9 | D8FDF6AD | D7B9DA25 | C45E4FDB |
| t = 78: | 5738D5E1 | 860D21CC | 681E6DF6 | D8FDF6AD | D7B9DA25 |
| t = 79: | 42541B35 | 5738D5E1 | 21834873 | 681E6DF6 | D8FDF6AD. |

Block 1 has been processed. The values of $\{H_i\}$ are

$H_0$ = 67452301 + 42541B35 = A9993E36
$H_1$ = EFCDAB89 + 5738D5E1 = 4706816A
$H_2$ = 98BADCFE + 21834873 = BA3E2571
$H_3$ = 10325476 + 681E6DF6 = 7850C26C
$H_4$ = C3D2E1F0 + D8FDF6AD = 9CD0D89D.

Message digest = A9993E36 4706816A BA3E2571 7850C26C 9CD0D89D

## APPENDIX B. A SECOND SAMPLE MESSAGE AND ITS MESSAGE DIGEST

This appendix is for informational purposes only and is not required to meet the standard.

Let the message be the binary-coded form (cf. Appendix A) of the ASCII string
"abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq".

Since each of the 56 characters is converted to 8 bits, the length of the message is l = 448. In step (a) of Section 4, we append "1". In step (b) we append 511 "0"s. In step (c) we append the 2-word representation of 448, i.e., hex 00000000 000001C0. This gives n = 2.

The initial hex values of $\{H_i\}$ are

$H_0$ = 67452301
$H_1$ = EFCDAB89
$H_2$ = 98BADCFE
$H_3$ = 10325476
$H_4$ = C3D2E1F0.

Start processing block 1. The words of block 1 are

W[0] = 61626364
W[1] = 62636465
W[2] = 63646566
W[3] = 64656667
W[4] = 65666768
W[5] = 66676869
W[6] = 6768696A
W[7] = 68696A6B
W[8] = 696A6B6C
W[9] = 6A6B6C6D
W[10] = 6B6C6D6E
W[11] = 6C6D6E6F
W[12] = 6D6E6F70
W[13] = 6E6F7071

W[14] = 80000000
W[15] = 00000000.

The hex values of A,B,C,D,E after pass t of the "for t = 0 to 79" loop (step (d) of Section 7 or step (c) of Section 8) are

| | A | B | C | D | E |
|---|---|---|---|---|---|
| t = 0: | 0116FC17 | 67452301 | 7BF36AE2 | 98BADCFE | 10325476 |
| t = 1: | EBF3B452 | 0116FC17 | 59D148C0 | 7BF36AE2 | 98BADCFE |
| t = 2: | 5109913A | EBF3B452 | C045BF05 | 59D148C0 | 7BF36AE2 |
| t = 3: | 2C4F6EAC | 5109913A | BAFCED14 | C045BF05 | 59D148C0 |
| t = 4: | 33F4AE5B | 2C4F6EAC | 9442644E | BAFCED14 | C045BF05 |
| t = 5: | 96B85189 | 33F4AE5B | 0B13DBAB | 9442644E | BAFCED14 |
| t = 6: | DB04CB58 | 96B85189 | CCFD2B96 | 0B13DBAB | 9442644E |
| t = 7: | 45833F0F | DB04CB58 | 65AE1462 | CCFD2B96 | 0B13DBAB |
| t = 8: | C565C35E | 45833F0F | 36C132D6 | 65AE1462 | CCFD2B96 |
| t = 9: | 6350AFDA | C565C35E | D160CFC3 | 36C132D6 | 65AE1462 |
| t = 10: | 8993EA77 | 6350AFDA | B15970D7 | D160CFC3 | 36C132D6 |
| t = 11: | E19ECAA2 | 8993EA77 | 98D42BF6 | B15970D7 | D160CFC3 |
| t = 12: | 8603481E | E19ECAA2 | E264FA9D | 98D42BF6 | B15970D7 |
| t = 13: | 32F94A85 | 8603481E | B867B2A8 | E264FA9D | 98D42BF6 |
| t = 14: | B2E7A8BE | 32F94A85 | A180D207 | B867B2A8 | E264FA9D |
| t = 15: | 42637E39 | B2E7A8BE | 4CBE52A1 | A180D207 | B867B2A8 |
| t = 16: | 6B068048 | 42637E39 | ACB9EA2F | 4CBE52A1 | A180D207 |
| t = 17: | 426B9C35 | 6B068048 | 5098DF8E | ACB9EA2F | 4CBE52A1 |
| t = 18: | 944B1BD1 | 426B9C35 | 1AC1A012 | 5098DF8E | ACB9EA2F |
| t = 19: | 6C445652 | 944B1BD1 | 509AE70D | 1AC1A012 | 5098DF8E |
| t = 20: | 95836DA5 | 6C445652 | 6512C6F4 | 509AE70D | 1AC1A012 |
| t = 21: | 09511177 | 95836DA5 | 9B111594 | 6512C6F4 | 509AE70D |
| t = 22: | E2B92DC4 | 09511177 | 6560DB69 | 9B111594 | 6512C6F4 |
| t = 23: | FD224575 | E2B92DC4 | C254445D | 6560DB69 | 9B111594 |
| t = 24: | EEB82D9A | FD224575 | 38AE4B71 | C254445D | 6560DB69 |
| t = 25: | 5A142C1A | EEB82D9A | 7F48915D | 38AE4B71 | C254445D |
| t = 26: | 2972F7C7 | 5A142C1A | BBAE0B66 | 7F48915D | 38AE4B71 |
| t = 27: | D526A644 | 2972F7C7 | 96850B06 | BBAE0B66 | 7F48915D |
| t = 28: | E1122421 | D526A644 | CA5CBDF1 | 96850B06 | BBAE0B66 |
| t = 29: | 05B457B2 | E1122421 | 3549A991 | CA5CBDF1 | 96850B06 |
| t = 30: | A9C84BEC | 05B457B2 | 78448908 | 3549A991 | CA5CBDF1 |
| t = 31: | 52E31F60 | A9C84BEC | 816D15EC | 78448908 | 3549A991 |
| t = 32: | 5AF3242C | 52E31F60 | 2A7212FB | 816D15EC | 78448908 |
| t = 33: | 31C756A9 | 5AF3242C | 14B8C7D8 | 2A7212FB | 816D15EC |
| t = 34: | E9AC987C | 31C756A9 | 16BCC90B | 14B8C7D8 | 2A7212FB |
| t = 35: | AB7C32EE | E9AC987C | 4C71D5AA | 16BCC90B | 14B8C7D8 |
| t = 36: | 5933FC99 | AB7C32EE | 3A6B261F | 4C71D5AA | 16BCC90B |
| t = 37: | 43F87AE9 | 5933FC99 | AADF0CBB | 3A6B261F | 4C71D5AA |
| t = 38: | 24957F22 | 43F87AE9 | 564CFF26 | AADF0CBB | 3A6B261F |
| t = 39: | ADEB7478 | 24957F22 | 50FE1EBA | 564CFF26 | AADF0CBB |
| t = 40: | D70E5010 | ADEB7478 | 89255FC8 | 50FE1EBA | 564CFF26 |
| t = 41: | 79BCFB08 | D70E5010 | 2B7ADD1E | 89255FC8 | 50FE1EBA |
| t = 42: | F9BCB8DE | 79BCFB08 | 35C39404 | 2B7ADD1E | 89255FC8 |
| t = 43: | 633E9561 | F9BCB8DE | 1E6F3EC2 | 35C39404 | 2B7ADD1E |
| t = 44: | 98C1EA64 | 633E9561 | BE6F2E37 | 1E6F3EC2 | 35C39404 |
| t = 45: | C6EA241E | 98C1EA64 | 58CFA558 | BE6F2E37 | 1E6F3EC2 |
| t = 46: | A2AD4F02 | C6EA241E | 26307A99 | 58CFA558 | BE6F2E37 |
| t = 47: | C8A69090 | A2AD4F02 | B1BA8907 | 26307A99 | 58CFA558 |
| t = 48: | 88341600 | C8A69090 | A8AB53C0 | B1BA8907 | 26307A99 |
| t = 49: | 7E846F58 | 88341600 | 3229A424 | A8AB53C0 | B1BA8907 |
| t = 50: | 86E358BA | 7E846F58 | 220D0580 | 3229A424 | A8AB53C0 |
| t = 51: | 8D2E76C8 | 86E358BA | 1FA11BD6 | 220D0580 | 3229A424 |
| t = 52: | CE892E10 | 8D2E76C8 | A1B8D62E | 1FA11BD6 | 220D0580 |

```
t = 53: EDEA95B1    CE892E10    234B9DB2    A1B8D62E    1FA11BD6
t = 54: 36D1230A    EDEA95B1    33A24B84    234B9DB2    A1B8D62E
t = 55: 776C3910    36D1230A    7B7AA56C    33A24B84    234B9DB2
t = 56: A681B723    776C3910    8DB448C2    7B7AA56C    33A24B84
t = 57: AC0A794F    A681B723    1DDB0E44    8DB448C2    7B7AA56C
t = 58: F03D3782    AC0A794F    E9A06DC8    1DDB0E44    8DB448C2
t = 59: 9EF775C3    F03D3782    EB029E53    E9A06DC8    1DDB0E44
t = 60: 36254B13    9EF775C3    BC0F4DE0    EB029E53    E9A06DC8
t = 61: 4080D4DC    36254B13    E7BDDD70    BC0F4DE0    EB029E53
t = 62: 2BFAF7A8    4080D4DC    CD8952C4    E7BDDD70    BC0F4DE0
t = 63: 513F9CA0    2BFAF7A8    10203537    CD8952C4    E7BDDD70
t = 64: E5895C81    513F9CA0    0AFEBDEA    10203537    CD8952C4
t = 65: 1037D2D5    E5895C81    144FE728    0AFEBDEA    10203537
t = 66: 14A82DA9    1037D2D5    79625720    144FE728    0AFEBDEA
t = 67: 6D17C9FD    14A82DA9    440DF4B5    79625720    144FE728
t = 68: 2C7B07BD    6D17C9FD    452A0B6A    440DF4B5    79625720
t = 69: FDF6EFFF    2C7B07BD    5B45F27F    452A0B6A    440DF4B5
t = 70: 112B96E3    FDF6EFFF    4B1EC1EF    5B45F27F    452A0B6A
t = 71: 84065712    112B96E3    FF7DBBFF    4B1EC1EF    5B45F27F
t = 72: AB89FB71    84065712    C44AE5B8    FF7DBBFF    4B1EC1EF
t = 73: C5210E35    AB89FB71    A10195C4    C44AE5B8    FF7DBBFF
t = 74: 352D9F4B    C5210E35    6AE27EDC    A10195C4    C44AE5B8
t = 75: 1A0E0E0A    352D9F4B    7148438D    6AE27EDC    A10195C4
t = 76: D0D47349    1A0E0E0A    CD4B67D2    7148438D    6AE27EDC
t = 77: AD38620D    D0D47349    86838382    CD4B67D2    7148438D
t = 78: D3AD7C25    AD38620D    74351CD2    86838382    CD4B67D2
t = 79: 8CE34517    D3AD7C25    6B4E1883    74351CD2    86838382.
```

Block 1 has been processed. The values of $\{H_i\}$ are

$H_0 = 67452301 + 8CE34517 = F4286818$
$H_1 = EFCDAB89 + D3AD7C25 = C37B27AE$
$H_2 = 98BADCFE + 6B4E1883 = 0408F581$
$H_3 = 10325476 + 74351CD2 = 84677148$
$H_4 = C3D2E1F0 + 86838382 = 4A566572.$

Start processing block 2. The words of block 2 are

W[0] = 00000000
W[1] = 00000000
W[2] = 00000000
W[3] = 00000000
W[4] = 00000000
W[5] = 00000000
W[6] = 00000000
W[7] = 00000000
W[8] = 00000000
W[9] = 00000000
W[10] = 00000000
W[11] = 00000000
W[12] = 00000000
W[13] = 00000000
W[14] = 00000000
W[15] = 000001C0.

The hex values of A,B,C,D,E after pass t of the for "t = 0 to 79" loop (step (d) of Section 7 or step (c) of Section 8) are

| | A | B | C | D | E |
|---|---|---|---|---|---|
| t = 0: | 2DF257E9 | F4286818 | B0DEC9EB | 0408F581 | 84677148 |
| t = 1: | 4D3DC58F | 2DF257E9 | 3D0A1A06 | B0DEC9EB | 0408F581 |
| t = 2: | C352BB05 | 4D3DC58F | 4B7C95FA | 3D0A1A06 | B0DEC9EB |
| t = 3: | EEF743C6 | C352BB05 | D34F7163 | 4B7C95FA | 3D0A1A06 |
| t = 4: | 41E34277 | EEF743C6 | 70D4AEC1 | D34F7163 | 4B7C95FA |
| t = 5: | 5443915C | 41E34277 | BBBDD0F1 | 70D4AEC1 | D34F7163 |
| t = 6: | E7FA0377 | 5443915C | D078D09D | BBBDD0F1 | 70D4AEC1 |
| t = 7: | C6946813 | E7FA0377 | 1510E457 | D078D09D | BBBDD0F1 |
| t = 8: | FDDE1DE1 | C6946813 | F9FE80DD | 1510E457 | D078D09D |
| t = 9: | B8538ACA | FDDE1DE1 | F1A51A04 | F9FE80DD | 1510E457 |
| t = 10: | 6BA94F63 | B8538ACA | 7F778778 | F1A51A04 | F9FE80DD |
| t = 11: | 43A2792F | 6BA94F63 | AE14E2B2 | 7F778778 | F1A51A04 |
| t = 12: | FECD7BBF | 43A2792F | DAEA53D8 | AE14E2B2 | 7F778778 |
| t = 13: | A2604CA8 | FECD7BBF | D0E89E4B | DAEA53D8 | AE14E2B2 |
| t = 14: | 258B0BAA | A2604CA8 | FFB35EEF | D0E89E4B | DAEA53D8 |
| t = 15: | D9772360 | 258B0BAA | 2898132A | FFB35EEF | D0E89E4B |
| t = 16: | 5507DB6E | D9772360 | 8962C2EA | 2898132A | FFB35EEF |
| t = 17: | A51B58BC | 5507DB6E | 365DC8D8 | 8962C2EA | 2898132A |
| t = 18: | C2EB709F | A51B58BC | 9541F6DB | 365DC8D8 | 8962C2EA |
| t = 19: | D8992153 | C2EB709F | 2946D62F | 9541F6DB | 365DC8D8 |
| t = 20: | 37482F5F | D8992153 | F0BADC27 | 2946D62F | 9541F6DB |
| t = 21: | EE8700BD | 37482F5F | F6264854 | F0BADC27 | 2946D62F |
| t = 22: | 9AD594B9 | EE8700BD | CDD20BD7 | F6264854 | F0BADC27 |
| t = 23: | 8FBAA5B9 | 9AD594B9 | 7BA1C02F | CDD20BD7 | F6264854 |
| t = 24: | 88FB5867 | 8FBAA5B9 | 66B5652E | 7BA1C02F | CDD20BD7 |
| t = 25: | EEC50521 | 88FB5867 | 63EEA96E | 66B5652E | 7BA1C02F |
| t = 26: | 50BCE434 | EEC50521 | E23ED619 | 63EEA96E | 66B5652E |
| t = 27: | 5C416DAF | 50BCE434 | 7BB14148 | E23ED619 | 63EEA96E |
| t = 28: | 2429BE5F | 5C416DAF | 142F390D | 7BB14148 | E23ED619 |
| t = 29: | 0A2FB108 | 2429BE5F | D7105B6B | 142F390D | 7BB14148 |
| t = 30: | 17986223 | 0A2FB108 | C90A6F97 | D7105B6B | 142F390D |
| t = 31: | 8A4AF384 | 17986223 | 028BEC42 | C90A6F97 | D7105B6B |
| t = 32: | 6B629993 | 8A4AF384 | C5E61888 | 028BEC42 | C90A6F97 |
| t = 33: | F15F04F3 | 6B629993 | 2292BCE1 | C5E61888 | 028BEC42 |
| t = 34: | 295CC25B | F15F04F3 | DAD8A664 | 2292BCE1 | C5E61888 |
| t = 35: | 696DA404 | 295CC25B | FC57C13C | DAD8A664 | 2292BCE1 |
| t = 36: | CEF5AE12 | 696DA404 | CA573096 | FC57C13C | DAD8A664 |
| t = 37: | 87D5B80C | CEF5AE12 | 1A5B6901 | CA573096 | FC57C13C |
| t = 38: | 84E2A5F2 | 87D5B80C | B3BD6B84 | 1A5B6901 | CA573096 |
| t = 39: | 03BB6310 | 84E2A5F2 | 21F56E03 | B3BD6B84 | 1A5B6901 |
| t = 40: | C2D8F75F | 03BB6310 | A138A97C | 21F56E03 | B3BD6B84 |
| t = 41: | BFB25768 | C2D8F75F | 00EED8C4 | A138A97C | 21F56E03 |
| t = 42: | 28589152 | BFB25768 | F0B63DD7 | 00EED8C4 | A138A97C |
| t = 43: | EC1D3D61 | 28589152 | 2FEC95DA | F0B63DD7 | 00EED8C4 |
| t = 44: | 3CAED7AF | EC1D3D61 | 8A162454 | 2FEC95DA | F0B63DD7 |
| t = 45: | C3D033EA | 3CAED7AF | 7B074F58 | 8A162454 | 2FEC95DA |
| t = 46: | 7316056A | C3D033EA | CF2BB5EB | 7B074F58 | 8A162454 |
| t = 47: | 46F93B68 | 7316056A | B0F40CFA | CF2BB5EB | 7B074F58 |
| t = 48: | DC8E7F26 | 46F93B68 | 9CC5815A | B0F40CFA | CF2BB5EB |
| t = 49: | 850D411C | DC8E7F26 | 11BE4EDA | 9CC5815A | B0F40CFA |
| t = 50: | 7E4672C0 | 850D411C | B7239FC9 | 11BE4EDA | 9CC5815A |
| t = 51: | 89FBD41D | 7E4672C0 | 21435047 | B7239FC9 | 11BE4EDA |
| t = 52: | 1797E228 | 89FBD41D | 1F919CB0 | 21435047 | B7239FC9 |
| t = 53: | 431D65BC | 1797E228 | 627EF507 | 1F919CB0 | 21435047 |
| t = 54: | 2BDBB8CB | 431D65BC | 05E5F88A | 627EF507 | 1F919CB0 |
| t = 55: | 6DA72E7F | 2BDBB8CB | 10C7596F | 05E5F88A | 627EF507 |

```
t = 56: A8495A9B    6DA72E7F    CAF6EE32    10C7596F    05E5F88A
t = 57: E785655A    A8495A9B    DB69CB9F    CAF6EE32    10C7596F
t = 58: 5B086C42    E785655A    EA1256A6    DB69CB9F    CAF6EE32
t = 59: A65818F7    5B086C42    B9E15956    EA1256A6    DB69CB9F
t = 60: 7AAB101B    A65818F7    96C21B10    B9E15956    EA1256A6
t = 61: 93614C9C    7AAB101B    E996063D    96C21B10    B9E15956
t = 62: F66D9BF4    93614C9C    DEAAC406    E996063D    96C21B10
t = 63: D504902B    F66D9BF4    24D85327    DEAAC406    E996063D
t = 64: 60A9DA62    D504902B    3D9B66FD    24D85327    DEAAC406
t = 65: 8B687819    60A9DA62    F541240A    3D9B66FD    24D85327
t = 66: 083E90C3    8B687819    982A7698    F541240A    3D9B66FD
t = 67: F6226BBF    083E90C3    62DA1E06    982A7698    F541240A
t = 68: 76C0563B    F6226BBF    C20FA430    62DA1E06    982A7698
t = 69: 989DD165    76C0563B    FD889AEF    C20FA430    62DA1E06
t = 70: 8B2C7573    989DD165    DDB0158E    FD889AEF    C20FA430
t = 71: AE1B8E7B    8B2C7573    66277459    DDB0158E    FD889AEF
t = 72: CA1840DE    AE1B8E7B    E2CB1D5C    66277459    DDB0158E
t = 73: 16F3BABB    CA1840DE    EB86E39E    E2CB1D5C    66277459
t = 74: D28D83AD    16F3BABB    B2861037    EB86E39E    E2CB1D5C
t = 75: 6BC02DFE    D28D83AD    C5BCEEAE    B2861037    EB86E39E
t = 76: D3A6E275    6BC02DFE    74A360EB    C5BCEEAE    B2861037
t = 77: DA955482    D3A6E275    9AF00B7F    74A360EB    C5BCEEAE
t = 78: 58C0AAC0    DA955482    74E9B89D    9AF00B7F    74A360EB
t = 79: 906FD62C    58C0AAC0    B6A55520    74E9B89D    9AF00B7F.
```

Block 2 has been processed. The values of $\{H_i\}$ are

$H_0$ = F4286818 + 906FD62C = 84983E44
$H_1$ = C37B27AE + 58C0AAC0 = 1C3BD26E
$H_2$ = 0408F581 + B6A55520 = BAAE4AA1
$H_3$ = 84677148 + 74E9B89D = F95129E5
$H_4$ = 4A566572 + 9AF00B7F = E54670F1.

Message digest = 84983E44 1C3BD26E BAAE4AA1 F95129E5 E54670F1

## APPENDIX C. A THIRD SAMPLE MESSAGE AND ITS MESSAGE DIGEST

This appendix is for informational purposes only and is not required to meet the standard.

Let the message be the binary-coded form of the ASCII string which consists of 1,000,000 repetitions of "a".

Message digest = 34AA973C D4C4DAA4 F61EEB2B DBAD2731 6534016F

---

**The Foreword, Abstract, and Key Words follow:**

FIPS PUB 180-1
FEDERAL INFORMATION
PROCESSING STANDARDS PUBLICATION

1995 April 17

U.S. DEPARTMENT OF COMMERCE/National Institute of Standards and Technology

# SECURE HASH STANDARD

U.S. DEPARTMENT OF COMMERCE, Ronald H. Brown, *Secretary*
National Institute of Standards and Technology, Arati Prabhakar, *Director*

## Foreword

The Federal Information Processing Standards Publication Series of the National Institute of Standards and Technology (NIST) is the official publication relating to standards and guidelines adopted and promulgated under the provisions of Section 111(d) of the Federal Property and Administrative Services Act of 1949 as amended by the Computer Security Act of 1987, Public Law 100-235. These mandates have given the Secretary of Commerce and NIST important responsibilities for improving the utilization and management of computers and related telecommunications systems in the Federal Government. The NIST, through its Computer Systems Laboratory, provides leadership, technical guidance, and coordination of Government efforts in the development of standards and guidelines in these areas.

Comments concerning Federal Information Processing Standards Publications are welcomed and should be addressed to the Director, Computer Systems Laboratory, National Institute of Standards and Technology, Gaithersburg, MD 20899.

James H. Burrows, *Director*
Computer Systems Laboratory

## Abstract

This standard specifies a Secure Hash Algorithm (SHA-1) which can be used to generate a condensed representation of a message called a message digest. The SHA-1 is required for use with the Digital Signature Algorithm (DSA) as specified in the Digital Signature Standard (DSS) and whenever a secure hash algorithm is required for Federal applications. The SHA-1 is used by both the transmitter and intended receiver of a message in computing and verifying a digital signature.

**Key words:** computer security; digital signatures; Federal Information Processing Standard (FIPS); hash algorithm.

---

Go Back to the Top.

Return to the FIPS
Home Page

---

# RFC1321



Network Working Group
Request for Comments: 1321

R. Rivest
MIT Laboratory for Computer Science
and RSA Data Security, Inc.
April 1992

The MD5 Message-Digest Algorithm

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard. Distribution of this memo is unlimited.

Acknowlegements

We would like to thank Don Coppersmith, Burt Kaliski, Ralph Merkle, David Chaum, and Noam Nisan for numerous helpful comments and suggestions.

Table of Contents



## 1. Executive Summary

This document describes the MD5 message-digest algorithm. The algorithm takes as input a message of arbitrary length and produces as output a 128-bit "fingerprint" or "message digest" of the input. It is conjectured that it is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given prespecified target message digest. The MD5 algorithm is intended for digital signature applications, where a large file must be "compressed" in a secure manner before being encrypted with a private (secret) key under a public-key cryptosystem such as RSA.

The MD5 algorithm is designed to be quite fast on 32-bit machines. In addition, the MD5 algorithm does not require any large substitution tables; the algorithm can be coded quite compactly.

The MD5 algorithm is an extension of the MD4 message-digest algorithm 1,2]. MD5 is slightly slower than MD4, but is more "conservative" in design. MD5 was designed because it was felt that MD4 was perhaps being adopted for use more quickly than justified by the existing critical review; because MD4 was designed to be exceptionally fast, it is "at the edge" in terms of risking successful cryptanalytic attack. MD5 backs off a bit, giving up a little in speed for a much greater likelihood of ultimate security. It incorporates some suggestions made by various reviewers, and contains additional optimizations. The MD5 algorithm is being placed in the public domain for review and possible adoption as a standard.

For OSI-based applications, MD5's object identifier is

```
md5 OBJECT IDENTIFIER ::=
  iso(1) member-body(2) US(840) rsadsi(113549) digestAlgorithm(2) 5}
```

In the X.509 type AlgorithmIdentifier [3], the parameters for MD5 should have type NULL.

## 2. Terminology and Notation

In this document a "word" is a 32-bit quantity and a "byte" is an eight-bit quantity. A sequence of bits can be interpreted in a natural manner as a sequence of bytes, where each consecutive group of eight bits is interpreted as a byte with the high-order (most significant) bit of each byte listed first. Similarly, a sequence of bytes can be interpreted as a sequence of 32-bit words, where each consecutive group of four bytes is interpreted as a word with the low-order (least significant) byte given first.

Let x_i denote "x sub i". If the subscript is an expression, we surround it in braces, as in x_{i+1}. Similarly, we use ^ for superscripts (exponentiation), so that x^i denotes x to the i-th power.

Let the symbol "+" denote addition of words (i.e., modulo-2^32 addition). Let X <<< s denote the 32-bit value obtained by circularly shifting (rotating) X left by s bit positions. Let not(X) denote the bit-wise complement of X, and let X v Y denote the bit-wise OR of X and Y. Let X xor Y denote the bit-wise XOR of X and Y, and let XY denote the bit-wise AND of X and Y.

## 3. MD5 Algorithm Description

We begin by supposing that we have a b-bit message as input, and that we wish to find its message digest. Here b is an arbitrary nonnegative integer; b may be zero, it need not be a multiple of eight, and it may be arbitrarily large. We imagine the bits of the message written down as follows:

```
m_0 m_1 ... m_{b-1}
```

The following five steps are performed to compute the message digest of the message.

### 3.1 Step 1. Append Padding Bits

The message is "padded" (extended) so that its length (in bits) is congruent to 448, modulo 512. That is, the message is extended so that it is just 64 bits shy of being a multiple of 512 bits long. Padding is always performed, even if the length of the message is already congruent to 448, modulo 512.

Padding is performed as follows: a single "1" bit is appended to the message, and then "0" bits are appended so that the length in bits of the padded message becomes congruent to 448, modulo 512. In all, at least one bit and at most 512 bits are appended.

### 3.2 Step 2. Append Length

A 64-bit representation of b (the length of the message before the padding bits were added) is appended to the result of the previous step. In the unlikely event that b is greater than 2^64, then only the low-order 64 bits of b are used. (These bits are appended as two 32-bit words and appended low-order word first in accordance with the previous conventions.)

At this point the resulting message (after padding with bits and with b) has a length that is an exact multiple of 512 bits. Equivalently, this message has a length that is an exact multiple of 16 (32-bit) words. Let M[0 ... N-1] denote the words of the resulting message, where N is a multiple of 16.

### 3.3 Step 3. Initialize MD Buffer

A four-word buffer (A,B,C,D) is used to compute the message digest. Here each of A, B, C, D is a 32-bit register. These registers are initialized to the following values in hexadecimal, low-order bytes first):

```
word A: 01 23 45 67
word B: 89 ab cd ef
word C: fe dc ba 98
word D: 76 54 32 10
```

### 3.4 Step 4. Process Message in 16-Word Blocks

We first define four auxiliary functions that each take as input three 32-bit words and produce as output one 32-bit word.

```
F(X,Y,Z) = XY v not(X) Z
G(X,Y,Z) = XZ v Y not(Z)
H(X,Y,Z) = X xor Y xor Z
I(X,Y,Z) = Y xor (X v not(Z))
```

In each bit position F acts as a conditional: if X then Y else Z. The function F could have been defined using + instead of v since XY and not(X)Z will never have 1's in the same bit position.) It is interesting to note that if the bits of X, Y, and Z are independent and unbiased, the each bit of F(X,Y,Z) will be independent and unbiased.

The functions G, H, and I are similar to the function F, in that they act in "bitwise parallel" to produce their output from the bits of X, Y, and Z, in such a manner that if the corresponding bits of X, Y, and Z are independent and unbiased, then each bit of G(X,Y,Z), H(X,Y,Z), and I(X,Y,Z) will be independent and unbiased. Note that the function H is the bit-wise "xor" or "parity" function of its inputs.

This step uses a 64-element table T[1 ... 64] constructed from the sine function. Let T[i] denote the i-th element of the table, which is equal to the integer part of 4294967296 times abs(sin(i)), where i is in radians. The elements of the table are given in the appendix.

Do the following:

```
/* Process each 16-word block. */
For i = 0 to N/16-1 do

  /* Copy block i into X. */
  For j = 0 to 15 do
    Set X[j] to M[i*16+j].
  end /* of loop on j */

  /* Save A as AA, B as BB, C as CC, and D as DD. */
  AA = A
  BB = B

  CC = C
  DD = D

  /* Round 1. */
  /* Let [abcd k s i] denote the operation
       a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */
  /* Do the following 16 operations. */
  [ABCD  0  7  1]  [DABC  1 12  2]  [CDAB  2 17  3]  [BCDA  3 22  4]
  [ABCD  4  7  5]  [DABC  5 12  6]  [CDAB  6 17  7]  [BCDA  7 22  8]
  [ABCD  8  7  9]  [DABC  9 12 10]  [CDAB 10 17 11]  [BCDA 11 22 12]
  [ABCD 12  7 13]  [DABC 13 12 14]  [CDAB 14 17 15]  [BCDA 15 22 16]

  /* Round 2. */
  /* Let [abcd k s i] denote the operation
       a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */
  /* Do the following 16 operations. */
  [ABCD  1  5 17]  [DABC  6  9 18]  [CDAB 11 14 19]  [BCDA  0 20 20]
  [ABCD  5  5 21]  [DABC 10  9 22]  [CDAB 15 14 23]  [BCDA  4 20 24]
  [ABCD  9  5 25]  [DABC 14  9 26]  [CDAB  3 14 27]  [BCDA  8 20 28]
  [ABCD 13  5 29]  [DABC  2  9 30]  [CDAB  7 14 31]  [BCDA 12 20 32]

  /* Round 3. */
  /* Let [abcd k s t] denote the operation
       a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */
  /* Do the following 16 operations. */
  [ABCD  5  4 33]  [DABC  8 11 34]  [CDAB 11 16 35]  [BCDA 14 23 36]
  [ABCD  1  4 37]  [DABC  4 11 38]  [CDAB  7 16 39]  [BCDA 10 23 40]
  [ABCD 13  4 41]  [DABC  0 11 42]  [CDAB  3 16 43]  [BCDA  6 23 44]
  [ABCD  9  4 45]  [DABC 12 11 46]  [CDAB 15 16 47]  [BCDA  2 23 48]

  /* Round 4. */
  /* Let [abcd k s t] denote the operation
       a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */
  /* Do the following 16 operations. */
  [ABCD  0  6 49]  [DABC  7 10 50]  [CDAB 14 15 51]  [BCDA  5 21 52]
  [ABCD 12  6 53]  [DABC  3 10 54]  [CDAB 10 15 55]  [BCDA  1 21 56]
  [ABCD  8  6 57]  [DABC 15 10 58]  [CDAB  6 15 59]  [BCDA 13 21 60]
  [ABCD  4  6 61]  [DABC 11 10 62]  [CDAB  2 15 63]  [BCDA  9 21 64]

  /* Then perform the following additions. (That is increment each
     of the four registers by the value it had before this block
     was started.) */
  A = A + AA
  B = B + BB
  C = C + CC
  D = D + DD

end /* of loop on i */
```

### 3.5 Step 5. Output

The message digest produced as output is A, B, C, D. That is, we

begin with the low-order byte of A, and end with the high-order byte of D.

This completes the description of MD5. A reference implementation in C is given in the appendix.

## 4. Summary

The MD5 message-digest algorithm is simple to implement, and provides a "fingerprint" or message digest of a message of arbitrary length. It is conjectured that the difficulty of coming up with two messages having the same message digest is on the order of 2^64 operations, and that the difficulty of coming up with any message having a given message digest is on the order of 2^128 operations. The MD5 algorithm has been carefully scrutinized for weaknesses. It is, however, a relatively new algorithm and further security analysis is of course justified, as is the case with any new proposal of this sort.

## 5. Differences Between MD4 and MD5

The following are the differences between MD4 and MD5:

1. A fourth round has been added.

2. Each step now has a unique additive constant.

3. The function g in round 2 was changed from (XY v XZ v YZ) to (XZ v Y not(Z)) to make g less symmetric.

4. Each step now adds in the result of the previous step. This promotes a faster "avalanche effect".

5. The order in which input words are accessed in rounds 2 and 3 is changed, to make these patterns less like each other.

6. The shift amounts in each round have been approximately optimized, to yield a faster "avalanche effect." The shifts in different rounds are distinct.

## References

[1] Rivest, R., "The MD4 Message Digest Algorithm", RFC 1320, MIT and RSA Data Security, Inc., April 1992.

[2] Rivest, R., "The MD4 message digest algorithm", in A.J. Menezes and S.A. Vanstone, editors, Advances in Cryptology - CRYPTO '90 Proceedings, pages 303-311, Springer-Verlag, 1991.

[3] CCITT Recommendation X.509 (1988), "The Directory - Authentication Framework."

## APPENDIX A - Reference Implementation

This appendix contains the following files taken from RSAREF: A Cryptographic Toolkit for Privacy-Enhanced Mail:

global.h -- global header file

md5.h -- header file for MD5

md5c.c -- source code for MD5

For more information on RSAREF, send email to <rsaref@rsa.com>.

The appendix also includes the following file:

mddriver.c -- test driver for MD2, MD4 and MD5

The driver compiles for MD5 by default but can compile for MD2 or MD4 if the symbol MD is defined on the C compiler command line as 2 or 4.

The implementation is portable and should work on many different plaforms. However, it is not difficult to optimize the implementation on particular platforms, an exercise left to the reader. For example, on "little-endian" platforms where the lowest-addressed byte in a 32-bit word is the least significant and there are no alignment restrictions, the call to Decode in MD5Transform can be replaced with a typecast.

## A.1 global.h

```
/* GLOBAL.H - RSAREF types and constants
 */

/* PROTOTYPES should be set to one if and only if the compiler supports
  function argument prototyping.
The following makes PROTOTYPES default to 0 if it has not already

  been defined with C compiler flags.
 */
#ifndef PROTOTYPES
#define PROTOTYPES 0
#endif

/* POINTER defines a generic pointer type */
typedef unsigned char *POINTER;

/* UINT2 defines a two byte word */
typedef unsigned short int UINT2;

/* UINT4 defines a four byte word */
typedef unsigned long int UINT4;

/* PROTO_LIST is defined depending on how PROTOTYPES is defined above.
If using PROTOTYPES, then PROTO_LIST returns the list, otherwise it
  returns an empty list.
 */
#if PROTOTYPES
#define PROTO_LIST(list) list
#else
#define PROTO_LIST(list) ()
#endif
```

## A.2 md5.h

```
/* MD5.H - header file for MD5C.C
 */

/* Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All
rights reserved.

License to copy and use this software is granted provided that it
is identified as the "RSA Data Security, Inc. MD5 Message-Digest
Algorithm" in all material mentioning or referencing this software
or this function.

License is also granted to make and use derivative works provided
that such works are identified as "derived from the RSA Data
```

Security, Inc. MD5 Message-Digest Algorithm" in all material mentioning or referencing the derived work.

RSA Data Security, Inc. makes no representations concerning either the merchantability of this software or the suitability of this software for any particular purpose. It is provided "as is" without express or implied warranty of any kind.

These notices must be retained in any copies of any part of this documentation and/or software.
*/

```
/* MD5 context. */
typedef struct {
  UINT4 state[4];                                   /* state (ABCD) */
  UINT4 count[2];        /* number of bits, modulo 2^64 (lsb first) */
  unsigned char buffer[64];                         /* input buffer */
} MD5_CTX;

void MD5Init PROTO_LIST ((MD5_CTX *));
void MD5Update PROTO_LIST
  ((MD5_CTX *, unsigned char *, unsigned int));
void MD5Final PROTO_LIST ((unsigned char [16], MD5_CTX *));
```

A.3 md5c.c

```
/* MD5C.C - RSA Data Security, Inc., MD5 message-digest algorithm
 */
```

/* Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All rights reserved.

License to copy and use this software is granted provided that it is identified as the "RSA Data Security, Inc. MD5 Message-Digest Algorithm" in all material mentioning or referencing this software or this function.

License is also granted to make and use derivative works provided that such works are identified as "derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm" in all material mentioning or referencing the derived work.

RSA Data Security, Inc. makes no representations concerning either the merchantability of this software or the suitability of this software for any particular purpose. It is provided "as is" without express or implied warranty of any kind.

These notices must be retained in any copies of any part of this documentation and/or software.
*/

```
#include "global.h"
#include "md5.h"

/* Constants for MD5Transform routine.
 */

#define S11 7
#define S12 12
#define S13 17
#define S14 22
#define S21 5
#define S22 9
#define S23 14
```

```
#define S24 20
#define S31 4
#define S32 11
#define S33 16
#define S34 23
#define S41 6
#define S42 10
#define S43 15
#define S44 21

static void MD5Transform PROTO_LIST ((UINT4 [4], unsigned char [64]));
static void Encode PROTO_LIST
  ((unsigned char *, UINT4 *, unsigned int));
static void Decode PROTO_LIST
  ((UINT4 *, unsigned char *, unsigned int));
static void MD5_memcpy PROTO_LIST ((POINTER, POINTER, unsigned int));
static void MD5_memset PROTO_LIST ((POINTER, int, unsigned int));

static unsigned char PADDING[64] = {
  0x80, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
};

/* F, G, H and I are basic MD5 functions.
 */
#define F(x, y, z) (((x) & (y)) | ((~x) & (z)))
#define G(x, y, z) (((x) & (z)) | ((y) & (~z)))
#define H(x, y, z) ((x) ^ (y) ^ (z))
#define I(x, y, z) ((y) ^ ((x) | (~z)))

/* ROTATE_LEFT rotates x left n bits.
 */
#define ROTATE_LEFT(x, n) (((x) << (n)) | ((x) >> (32-(n))))

/* FF, GG, HH, and II transformations for rounds 1, 2, 3, and 4.
Rotation is separate from addition to prevent recomputation.
 */
#define FF(a, b, c, d, x, s, ac) { \
 (a) += F ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \

 (a) += (b); \
  }
#define GG(a, b, c, d, x, s, ac) { \
 (a) += G ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
 (a) += (b); \
  }
#define HH(a, b, c, d, x, s, ac) { \
 (a) += H ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
 (a) += (b); \
  }
#define II(a, b, c, d, x, s, ac) { \
 (a) += I ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
 (a) += (b); \
  }

/* MD5 initialization. Begins an MD5 operation, writing a new context.
 */
void MD5Init (context)
MD5_CTX *context;                                        /* context */
```

```
{
  context->count[0] = context->count[1] = 0;
  /* Load magic initialization constants.
*/
  context->state[0] = 0x67452301;
  context->state[1] = 0xefcdab89;
  context->state[2] = 0x98badcfe;
  context->state[3] = 0x10325476;
}

/* MD5 block update operation. Continues an MD5 message-digest
  operation, processing another message block, and updating the
  context.
 */
void MD5Update (context, input, inputLen)
MD5_CTX *context;                                        /* context */
unsigned char *input;                                /* input block */
unsigned int inputLen;                     /* length of input block */
{
  unsigned int i, index, partLen;

  /* Compute number of bytes mod 64 */
  index = (unsigned int)((context->count[0] >> 3) & 0x3F);

  /* Update number of bits */
  if ((context->count[0] += ((UINT4)inputLen << 3))

   < ((UINT4)inputLen << 3))
 context->count[1]++;
  context->count[1] += ((UINT4)inputLen >> 29);

  partLen = 64 - index;

  /* Transform as many times as possible.
*/
  if (inputLen >= partLen) {
 MD5_memcpy
   ((POINTER)&context->buffer[index], (POINTER)input, partLen);
 MD5Transform (context->state, context->buffer);

 for (i = partLen; i + 63 < inputLen; i += 64)
   MD5Transform (context->state, &input[i]);

 index = 0;
  }
  else
 i = 0;

  /* Buffer remaining input */
  MD5_memcpy
 ((POINTER)&context->buffer[index], (POINTER)&input[i],
  inputLen-i);
}

/* MD5 finalization. Ends an MD5 message-digest operation, writing the
  the message digest and zeroizing the context.
 */
void MD5Final (digest, context)
unsigned char digest[16];                         /* message digest */
MD5_CTX *context;                                        /* context */
{
  unsigned char bits[8];
  unsigned int index, padLen;
```

```
    /* Save number of bits */
    Encode (bits, context->count, 8);

    /* Pad out to 56 mod 64.
  */
    index = (unsigned int)((context->count[0] >> 3) & 0x3f);
    padLen = (index < 56) ? (56 - index) : (120 - index);
    MD5Update (context, PADDING, padLen);

    /* Append length (before padding) */
    MD5Update (context, bits, 8);

    /* Store state in digest */
    Encode (digest, context->state, 16);

    /* Zeroize sensitive information.
  */
    MD5_memset ((POINTER)context, 0, sizeof (*context));
  }

  /* MD5 basic transformation. Transforms state based on block.
   */
  static void MD5Transform (state, block)
  UINT4 state[4];
  unsigned char block[64];
  {
    UINT4 a = state[0], b = state[1], c = state[2], d = state[3], x[16];

    Decode (x, block, 64);

    /* Round 1 */
    FF (a, b, c, d, x[ 0], S11, 0xd76aa478); /* 1 */
    FF (d, a, b, c, x[ 1], S12, 0xe8c7b756); /* 2 */
    FF (c, d, a, b, x[ 2], S13, 0x242070db); /* 3 */
    FF (b, c, d, a, x[ 3], S14, 0xc1bdceee); /* 4 */
    FF (a, b, c, d, x[ 4], S11, 0xf57c0faf); /* 5 */
    FF (d, a, b, c, x[ 5], S12, 0x4787c62a); /* 6 */
    FF (c, d, a, b, x[ 6], S13, 0xa8304613); /* 7 */
    FF (b, c, d, a, x[ 7], S14, 0xfd469501); /* 8 */
    FF (a, b, c, d, x[ 8], S11, 0x698098d8); /* 9 */
    FF (d, a, b, c, x[ 9], S12, 0x8b44f7af); /* 10 */
    FF (c, d, a, b, x[10], S13, 0xffff5bb1); /* 11 */
    FF (b, c, d, a, x[11], S14, 0x895cd7be); /* 12 */
    FF (a, b, c, d, x[12], S11, 0x6b901122); /* 13 */
    FF (d, a, b, c, x[13], S12, 0xfd987193); /* 14 */
    FF (c, d, a, b, x[14], S13, 0xa679438e); /* 15 */
    FF (b, c, d, a, x[15], S14, 0x49b40821); /* 16 */

   /* Round 2 */
    GG (a, b, c, d, x[ 1], S21, 0xf61e2562); /* 17 */
    GG (d, a, b, c, x[ 6], S22, 0xc040b340); /* 18 */
    GG (c, d, a, b, x[11], S23, 0x265e5a51); /* 19 */
    GG (b, c, d, a, x[ 0], S24, 0xe9b6c7aa); /* 20 */
    GG (a, b, c, d, x[ 5], S21, 0xd62f105d); /* 21 */
    GG (d, a, b, c, x[10], S22,  0x2441453); /* 22 */
    GG (c, d, a, b, x[15], S23, 0xd8a1e681); /* 23 */
    GG (b, c, d, a, x[ 4], S24, 0xe7d3fbc8); /* 24 */
    GG (a, b, c, d, x[ 9], S21, 0x21e1cde6); /* 25 */
    GG (d, a, b, c, x[14], S22, 0xc33707d6); /* 26 */
    GG (c, d, a, b, x[ 3], S23, 0xf4d50d87); /* 27 */

    GG (b, c, d, a, x[ 8], S24, 0x455a14ed); /* 28 */
    GG (a, b, c, d, x[13], S21, 0xa9e3e905); /* 29 */
    GG (d, a, b, c, x[ 2], S22, 0xfcefa3f8); /* 30 */
```

```
    GG (c, d, a, b, x[ 7], S23, 0x676f02d9); /* 31 */
    GG (b, c, d, a, x[12], S24, 0x8d2a4c8a); /* 32 */

    /* Round 3 */
    HH (a, b, c, d, x[ 5], S31, 0xfffa3942); /* 33 */
    HH (d, a, b, c, x[ 8], S32, 0x8771f681); /* 34 */
    HH (c, d, a, b, x[11], S33, 0x6d9d6122); /* 35 */
    HH (b, c, d, a, x[14], S34, 0xfde5380c); /* 36 */
    HH (a, b, c, d, x[ 1], S31, 0xa4beea44); /* 37 */
    HH (d, a, b, c, x[ 4], S32, 0x4bdecfa9); /* 38 */
    HH (c, d, a, b, x[ 7], S33, 0xf6bb4b60); /* 39 */
    HH (b, c, d, a, x[10], S34, 0xbebfbc70); /* 40 */
    HH (a, b, c, d, x[13], S31, 0x289b7ec6); /* 41 */
    HH (d, a, b, c, x[ 0], S32, 0xeaa127fa); /* 42 */
    HH (c, d, a, b, x[ 3], S33, 0xd4ef3085); /* 43 */
    HH (b, c, d, a, x[ 6], S34,  0x4881d05); /* 44 */
    HH (a, b, c, d, x[ 9], S31, 0xd9d4d039); /* 45 */
    HH (d, a, b, c, x[12], S32, 0xe6db99e5); /* 46 */
    HH (c, d, a, b, x[15], S33, 0x1fa27cf8); /* 47 */
    HH (b, c, d, a, x[ 2], S34, 0xc4ac5665); /* 48 */

    /* Round 4 */
    II (a, b, c, d, x[ 0], S41, 0xf4292244); /* 49 */
    II (d, a, b, c, x[ 7], S42, 0x432aff97); /* 50 */
    II (c, d, a, b, x[14], S43, 0xab9423a7); /* 51 */
    II (b, c, d, a, x[ 5], S44, 0xfc93a039); /* 52 */
    II (a, b, c, d, x[12], S41, 0x655b59c3); /* 53 */
    II (d, a, b, c, x[ 3], S42, 0x8f0ccc92); /* 54 */
    II (c, d, a, b, x[10], S43, 0xffeff47d); /* 55 */
    II (b, c, d, a, x[ 1], S44, 0x85845dd1); /* 56 */
    II (a, b, c, d, x[ 8], S41, 0x6fa87e4f); /* 57 */
    II (d, a, b, c, x[15], S42, 0xfe2ce6e0); /* 58 */
    II (c, d, a, b, x[ 6], S43, 0xa3014314); /* 59 */
    II (b, c, d, a, x[13], S44, 0x4e0811a1); /* 60 */
    II (a, b, c, d, x[ 4], S41, 0xf7537e82); /* 61 */
    II (d, a, b, c, x[11], S42, 0xbd3af235); /* 62 */
    II (c, d, a, b, x[ 2], S43, 0x2ad7d2bb); /* 63 */
    II (b, c, d, a, x[ 9], S44, 0xeb86d391); /* 64 */

    state[0] += a;
    state[1] += b;
    state[2] += c;
    state[3] += d;

    /* Zeroize sensitive information.

  */
    MD5_memset ((POINTER)x, 0, sizeof (x));
  }

  /* Encodes input (UINT4) into output (unsigned char). Assumes len is
    a multiple of 4.
   */
  static void Encode (output, input, len)
  unsigned char *output;
  UINT4 *input;
  unsigned int len;
  {
    unsigned int i, j;

    for (i = 0, j = 0; j < len; i++, j += 4) {
   output[j] = (unsigned char)(input[i] & 0xff);
   output[j+1] = (unsigned char)((input[i] >> 8) & 0xff);
   output[j+2] = (unsigned char)((input[i] >> 16) & 0xff);
```

```
    output[j+3] = (unsigned char)((input[i] >> 24) & 0xff);
  }
}

/* Decodes input (unsigned char) into output (UINT4). Assumes len is
  a multiple of 4.
 */
static void Decode (output, input, len)
UINT4 *output;
unsigned char *input;
unsigned int len;
{
  unsigned int i, j;

  for (i = 0, j = 0; j < len; i++, j += 4)
 output[i] = ((UINT4)input[j]) | (((UINT4)input[j+1]) << 8) |
   (((UINT4)input[j+2]) << 16) | (((UINT4)input[j+3]) << 24);
}

/* Note: Replace "for loop" with standard memcpy if possible.
 */

static void MD5_memcpy (output, input, len)
POINTER output;
POINTER input;
unsigned int len;
{
  unsigned int i;

  for (i = 0; i < len; i++)

 output[i] = input[i];
}

/* Note: Replace "for loop" with standard memset if possible.
 */
static void MD5_memset (output, value, len)
POINTER output;
int value;
unsigned int len;
{
  unsigned int i;

  for (i = 0; i < len; i++)
 ((char *)output)[i] = (char)value;
}
```

A.4 mddriver.c

```
/* MDDRIVER.C - test driver for MD2, MD4 and MD5
 */

/* Copyright (C) 1990-2, RSA Data Security, Inc. Created 1990. All
rights reserved.

RSA Data Security, Inc. makes no representations concerning either
the merchantability of this software or the suitability of this
software for any particular purpose. It is provided "as is"
without express or implied warranty of any kind.

These notices must be retained in any copies of any part of this
documentation and/or software.
 */
```

```
/* The following makes MD default to MD5 if it has not already been
  defined with C compiler flags.
 */
#ifndef MD
#define MD MD5
#endif

#include <stdio.h>
#include <time.h>
#include <string.h>
#include "global.h"
#if MD == 2
#include "md2.h"
#endif
#if MD == 4

#include "md4.h"
#endif
#if MD == 5
#include "md5.h"
#endif

/* Length of test block, number of test blocks.
 */
#define TEST_BLOCK_LEN 1000
#define TEST_BLOCK_COUNT 1000

static void MDString PROTO_LIST ((char *));
static void MDTimeTrial PROTO_LIST ((void));
static void MDTestSuite PROTO_LIST ((void));
static void MDFile PROTO_LIST ((char *));
static void MDFilter PROTO_LIST ((void));
static void MDPrint PROTO_LIST ((unsigned char [16]));

#if MD == 2
#define MD_CTX MD2_CTX
#define MDInit MD2Init
#define MDUpdate MD2Update
#define MDFinal MD2Final
#endif
#if MD == 4
#define MD_CTX MD4_CTX
#define MDInit MD4Init
#define MDUpdate MD4Update
#define MDFinal MD4Final
#endif
#if MD == 5
#define MD_CTX MD5_CTX
#define MDInit MD5Init
#define MDUpdate MD5Update
#define MDFinal MD5Final
#endif

/* Main driver.

Arguments (may be any combination):
  -sstring - digests string
  -t       - runs time trial
  -x       - runs test script
  filename - digests file
  (none)   - digests standard input
 */
int main (argc, argv)
int argc;
```

```
char *argv[];
{
  int i;

  if (argc > 1)
 for (i = 1; i < argc; i++)
   if (argv[i][0] == '-' && argv[i][1] == 's')
     MDString (argv[i] + 2);
   else if (strcmp (argv[i], "-t") == 0)
     MDTimeTrial ();
   else if (strcmp (argv[i], "-x") == 0)
     MDTestSuite ();
   else
     MDFile (argv[i]);
  else
 MDFilter ();

  return (0);
}

/* Digests a string and prints the result.
 */
static void MDString (string)
char *string;
{
  MD_CTX context;
  unsigned char digest[16];
  unsigned int len = strlen (string);

  MDInit (&context);
  MDUpdate (&context, string, len);
  MDFinal (digest, &context);

  printf ("MD%d (\"%s\") = ", MD, string);
  MDPrint (digest);
  printf ("\n");
}

/* Measures the time to digest TEST_BLOCK_COUNT TEST_BLOCK_LEN-byte
  blocks.
 */
static void MDTimeTrial ()
{
  MD_CTX context;
  time_t endTime, startTime;
  unsigned char block[TEST_BLOCK_LEN], digest[16];
  unsigned int i;

  printf
 ("MD%d time trial. Digesting %d %d-byte blocks ...", MD,
  TEST_BLOCK_LEN, TEST_BLOCK_COUNT);

  /* Initialize block */
  for (i = 0; i < TEST_BLOCK_LEN; i++)
 block[i] = (unsigned char)(i & 0xff);

  /* Start timer */
  time (&startTime);

  /* Digest blocks */
  MDInit (&context);
  for (i = 0; i < TEST_BLOCK_COUNT; i++)
 MDUpdate (&context, block, TEST_BLOCK_LEN);
```

```
  MDFinal (digest, &context);

  /* Stop timer */
  time (&endTime);

  printf (" done\n");
  printf ("Digest = ");
  MDPrint (digest);
  printf ("\nTime = %ld seconds\n", (long)(endTime-startTime));
  printf
 ("Speed = %ld bytes/second\n",
  (long)TEST_BLOCK_LEN * (long)TEST_BLOCK_COUNT/(endTime-startTime));
}

/* Digests a reference suite of strings and prints the results.
 */
static void MDTestSuite ()
{
  printf ("MD%d test suite:\n", MD);

  MDString ("");
  MDString ("a");
  MDString ("abc");
  MDString ("message digest");
  MDString ("abcdefghijklmnopqrstuvwxyz");
  MDString
 ("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789");
  MDString
 ("12345678901234567890123456789012345678901\
2345678901234567890123456789012345678901234567890");
}

/* Digests a file and prints the result.

 */
static void MDFile (filename)
char *filename;
{
  FILE *file;
  MD_CTX context;
  int len;
  unsigned char buffer[1024], digest[16];

  if ((file = fopen (filename, "rb")) == NULL)
 printf ("%s can't be opened\n", filename);

  else {
 MDInit (&context);
 while (len = fread (buffer, 1, 1024, file))
   MDUpdate (&context, buffer, len);
 MDFinal (digest, &context);

 fclose (file);

 printf ("MD%d (%s) = ", MD, filename);
 MDPrint (digest);
 printf ("\n");
  }
}

/* Digests the standard input and prints the result.
 */
static void MDFilter ()
{
```

```
  MD_CTX context;
  int len;
  unsigned char buffer[16], digest[16];

  MDInit (&context);
  while (len = fread (buffer, 1, 16, stdin))
 MDUpdate (&context, buffer, len);
  MDFinal (digest, &context);

  MDPrint (digest);
  printf ("\n");
}

/* Prints a message digest in hexadecimal.
 */
static void MDPrint (digest)
unsigned char digest[16];
{

  unsigned int i;

  for (i = 0; i < 16; i++)
 printf ("%02x", digest[i]);
}
```

A.5 Test suite

The MD5 test suite (driver option "-x") should print the following results:

```
MD5 test suite:
MD5 ("") = d41d8cd98f00b204e9800998ecf8427e
MD5 ("a") = 0cc175b9c0f1b6a831c399e269772661
MD5 ("abc") = 900150983cd24fb0d6963f7d28e17f72
MD5 ("message digest") = f96b697d7cb7938d525a2f31aaf161d0
MD5 ("abcdefghijklmnopqrstuvwxyz") = c3fcd3d76192e4007dfb496cca67e13b
MD5 ("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789") =
d174ab98d277d9f5a5611c2c9f419d9f
MD5 ("12345678901234567890123456789012345678901234567890123456789012345678
78901234567890") = 57edf4a22be3c955ac49da2e2107b67a
```

Security Considerations

The level of security discussed in this memo is considered to be sufficient for implementing very high security hybrid digital-signature schemes based on MD5 and a public-key cryptosystem.

Author's Address

Ronald L. Rivest
Massachusetts Institute of Technology
Laboratory for Computer Science
NE43-324
545 Technology Square
Cambridge, MA 02139-1986

Phone: (617) 253-5880
EMail: rivest@theory.lcs.mit.edu